

Combining structural and symbolic methods for the verification of concurrent systems

Jordi Cortadella*
Department of Software
Universitat Politècnica de Catalunya
Barcelona, Spain

Abstract

The contributions during the last few years on the structural theory of Petri nets can now be applied to formal verification. The structural theory provides methods to find efficient encoding schemes for symbolic representations of the reachable markings. It also provides approximations of the state space that allow to alleviate many bottlenecks in the calculation of the reachability set by breadth or depth first search algorithms.

The paper reviews some of the results on the structural theory and explains how they can be incorporated in a model-checking verification framework for concurrent systems.

1 Introduction

Formal verification of concurrent systems suffers from the state explosion problem. The number of states of a system can grow exponentially in the number of subsystems.

One major challenge in the ongoing research on verification is to significantly increase the size of the systems that can be verified. The progress achieved by symbolic model-checking techniques have approached the verification domain to practical-sized systems. However, there are still serious limitations of time and memory for many cases.

We discuss here several techniques for the verification of systems modeled with Petri nets [16]. For many years, Petri nets have been the target of many researchers and different theoretical results have emerged. These results can now be used to alleviate some of the verification bottlenecks.

We consider the verification of concurrent systems using temporal logics such as linear temporal logic (LTL), computation tree logic (CTL) or μ -calculus [1]. Typically, temporal logic formulae can describe *state* and *path* properties. An example of state property

is “*at most one writer has access to the database*”. This is a property that can be checked locally for each state of the system. On the other hand, the property “*every request will be eventually acknowledged*” is a path property that must be checked for all possible sequences of events of the system.

Verifying a property often requires the exploration of the state space. To reduce the complexity of such exploration, approaches going to opposite directions can be devised, namely,

- by reducing the state space while preserving the properties that must be verified or
- by enlarging the state space, making verification conservative (no false positives) but reducing the symbolic representation of the state space.

The main contribution of this work is to show how structural and symbolic techniques can be combined in the same verification framework.

The techniques we will discuss can be classified according to their effect on the calculation and representation of the state space:

- State reduction and abstraction techniques.
- Symbolic representation of the state space.
- Approximations of the state space.

We assume the reader to be familiar with Petri nets and symbolic model checking techniques. We refer the reader to [12, 11] for a basic background on these topics.

2 State reduction and abstraction techniques

Partial-order reduction techniques have been proposed to reduce the state space generated by concurrent systems [18, 15]. Intuitively, the main observation of these methods relies on the fact that concurrent

*This work has been funded by CICYT TIC 95-0419

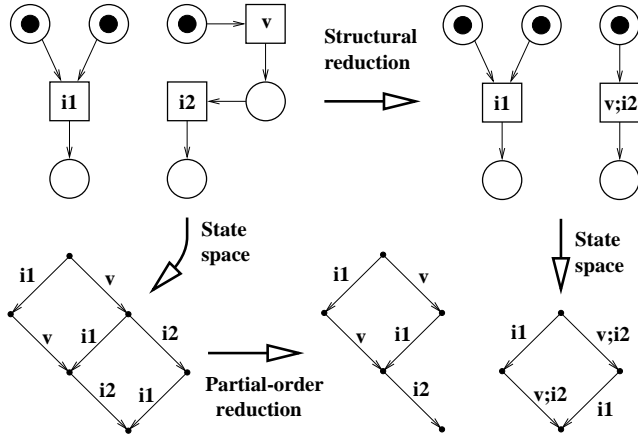


Figure 1: State reduction and abstraction techniques

events are modeled by a set of sequences executing all possible interleavings. When the execution order is irrelevant for the properties that must be verified, it is enough to choose one of them to preserve the behavioral skeleton of the system without losing accuracy in the verification task.

An example is illustrated in Figure 1. Assume that i_1 and i_2 denote *invisible* events from the point of view of the properties that must be verified. Clearly, the transitions labeled with i_1 and i_2 are independent since they do not share any input/output place. Under such conditions, i_1 and i_2 will occur concurrently when enabled and any of the sequences $i_1; i_2$ or $i_2; i_1$ could be produced. Partial-order reduction techniques would choose only one of them, thus resulting in a reduction of the state space. In practice, this technique can be applied by reducing the set of enabled events explored at each state when building the reachability set of the system with breadth or depth first search algorithms [15].

When the formalism to model the system is a Petri net, reduction rules to transform the net into a simpler one that preserves the relevant properties can be applied [12, 17]. The example of Figure 1 illustrates one of such rules. Transitions labeled with v and i_2 represent a sequence of these two events. A reduction ruled called “*fusion of series transitions*” can be applied and obtain a new transition that abstracts the behavior of both events into a single event labeled $v; i_2$. Such type of rules can be used to automatically remove invisible actions (e.g. i_2 could be removed from the label $v; i_2$) or to derive a symbolic representation of the state space in a hierarchical manner [14].

3 Symbolic representation of the state space

Ordered binary decision diagrams (OBDDs) [2] have emerged as an efficient form to represent boolean functions and have provided a crucial toolbox for verification systems based on symbolic model checking techniques [11].

Petri nets present a structure appropriate for boolean encoding. If we consider a safe Petri net¹, the state of each place can be encoded by one boolean variable. Thus, the reachability set of the Petri net can be represented by a boolean characteristic function and manipulated by boolean operations [14].

Figure 2 depicts a safe Petri net. Its reachability set is represented by the state graph at the left of the figure (8 states). Each state is labeled with the indices of the marked places. The set of reachable markings can be characterized by the boolean function²

$$S = (s_1 \oplus s_4)(s_2 \oplus s_3)(s_5 \oplus s_8)(s_6 \oplus s_7) \\ ((s_1 \oplus s_2) \Leftrightarrow (s_5 \oplus s_6)) \quad (1)$$

that can be efficiently represented by an OBDD.

Traversal algorithms for building the reachability set of the Petri net from its initial marking can be efficiently implemented by using boolean operations [4, 3]. In particular, if m_0 is the initial marking of a net N , the reachability set S can be obtained by computing the least fix point of the following recurrence:

$$S_0 = \{m_0\} \\ S_{i+1} = S_i \cup \text{Image}(N, S_i) \quad (2)$$

where *Image* is a function that returns the states reachable from S_i in one step. In the example, $\text{Image}(N, \{[1256]\}) = \{[3456], [1278]\}$.

The efficiency of OBDD-based methods manipulating sets of markings has been shown by different authors. As an example, [14] shows how a Petri net modeling the dining philosophers paradigm can represent the reachability set for 28 philosophers (4.8×10^{18} markings) with an OBDD of about 10^3 nodes.

4 Approximations of the state space

The exact exploration of the state space can be a tedious task, even for symbolic representations of such space. It is well known that, although the final symbolic representation of the state space can be small, traversal algorithms, such as the one defined by the

¹the method can be easily extended to k -bounded Petri nets [14].

² \oplus and \Leftrightarrow denote XOR and XNOR operations respectively.

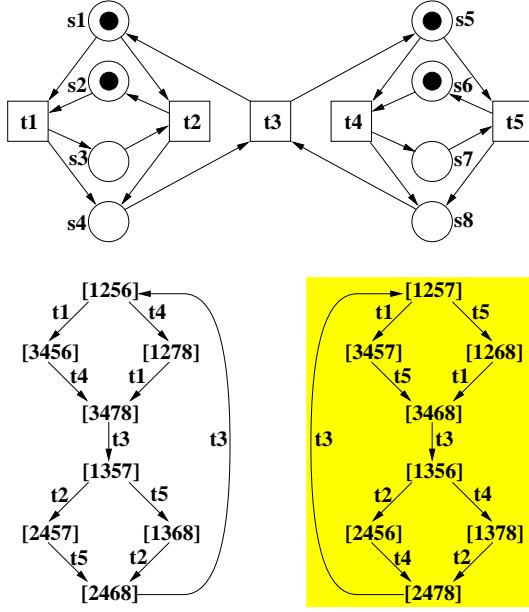


Figure 2: Safe Petri net and its reachability set (example from [5]).

recurrence (2), often suffer from the size of the representation of S_i at intermediate steps of the exploration. This phenomenon may cause the exploration to become impractical.

Here we review some methods from the structural theory of Petri nets that can alleviate most of these problems. In particular, they can help to find

- an efficient encoding of the state space
- conservative approximations of the state space without executing search algorithms
- successive refinements of the state space approximation

4.1 Efficient encoding

Assume that it is possible to identify a set of safe places $P' = \{p_1, \dots, p_n\}$ that are not pairwise concurrent, i.e. no pair of places can be marked simultaneously. Thus, at most one token will mark the places in P' at any reachable marking of the net. Therefore, the places in P' can only be in $n + 1$ different states. In the case that some place is always marked, only n states are possible. Under this constraint, the state of the places in P' can be encoded with $\lceil \log_2(n + 1) \rceil$ (or $\lceil \log_2 n \rceil$ if always marked) boolean variables [13].

Identifying places that are not concurrent can be conservatively performed by computing the *structural*

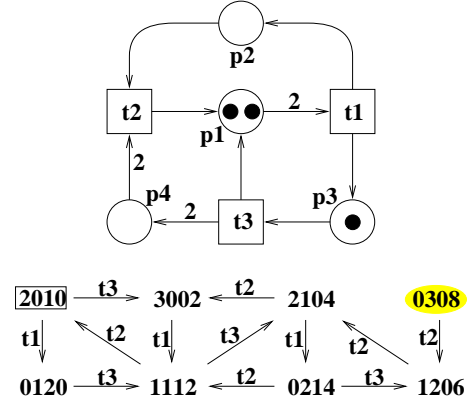


Figure 3: Potentially reachable markings (example from [12]).

concurrency relation [9] that gives necessary conditions for two places not to be concurrent. A complementary way is the calculation of *state machines* initially marked with one token. State machines correspond to place invariants that can be obtained by using algebraic methods [7, 12].

Let us illustrate this feature with the example of Figure 2. There are four place invariants that define state machines of the net. They correspond to the sets of places $\{s_1, s_4\}$, $\{s_2, s_3\}$, $\{s_5, s_8\}$ and $\{s_6, s_7\}$. In all cases, there is always one place in the set that is marked in the state space. This property can be structurally deduced from the fact that they define state machines of the net. Thus, the state of each set of places can be encoded with one boolean variable. Let us call these variables x_1, \dots, x_4 respectively, i.e. $x_1 \Rightarrow s_1 \bar{s}_4$, $\bar{x}_1 \Rightarrow \bar{s}_1 s_4$, $x_2 \Rightarrow s_2 \bar{s}_3$, and so on. The set of reachable states can now be characterized by the boolean equation

$$S = (x_1 \oplus x_2) \Leftrightarrow (x_3 \oplus x_4) \quad (3)$$

which is much simpler than (1).

4.2 Conservative approximations

The structural theory of Petri nets provides efficient mechanisms to derive the so-called *potentially reachable state space*, that corresponds to the set of markings that fulfil the *state equation* of the Petri net [12]. The state equation gives a superset of the state space since any reachable marking fulfils the state equation, but not vice versa.

Figure 3 depicts a Petri net and its potentially reachable state space. The states are labeled with four digits that correspond to the token count of $p_1 \dots p_4$ respectively (the initial marking is 2010). We can ob-

serve than one of the markings fulfilling the state equation, 0308, is not reachable from the initial marking.

Using a superset of the state space results in some limitations of the predicates that can be verified. Thus, properties that hold *for all* states in a set (*universal quantifiers*) also hold for any subset of states. However properties that hold when *there exists* some state with a specific characteristic in a set (*existential quantifiers*) only hold for supersets. Therefore, the potentially reachable state space provides a conservative method to verify properties without existential quantifiers. On the other hand, subsets of the reachable state space can be useful for conservative verification of properties with existential quantifiers.

4.3 Refinements of the state space approximation

We discuss here two approaches to derive successive refinements of the state space.

Backward state elimination

The potentially reachable state space also provides a starting point to calculate the exact state space using backward state elimination. If we call \hat{S}_0 the potentially reachable state space of a net and S the reachable state space, the following recurrence gives successive subsets $S \subseteq \hat{S}_{i+1} \subseteq \hat{S}_i \subseteq \hat{S}_0$:

$$\hat{S}_{i+1} = \text{Image}(N, \hat{S}_i) \cup \{m_0\} \quad (4)$$

At each step of the recurrence, all those states that are not reachable from any of the states of \hat{S}_i are eliminated in \hat{S}_{i+1} , except the state corresponding to the initial marking m_0 .

In the example of Figure 3, S can be obtained from \hat{S}_0 by applying the previous recurrence only once, thus eliminating the state 0308 from the reachable set. Unfortunately, a fix point does not always guarantee the exact state space. This is illustrated by the example of Figure 2. The shadowed states fulfil the state equation and, therefore, belong to \hat{S}_0 . However the backward state elimination cannot remove any state since all states are reachable from some state of the set.

In the worst case, still any \hat{S}_i gives an initial set of unreachable states. This knowledge can be crucial to make the state exploration much more efficient by taking the unreachable states as “don’t cares” of the boolean functions used to represent the transition relation and the reachability set of the net [3].

Modulo-invariants

Desel et al. [5] introduced modulo-invariants as a generalization of the concept of place-invariants. The interesting property of modulo-invariants is that a basis can be calculated in polynomial time from the *incidence matrix* of the net by obtaining its Smith Normal Form [8]. Besides providing the conventional place invariants that can be used to encode the token count of the places, as explained in section 4.1, they also provide extra information to prune the potentially reachable state space.

Let us take again the example of Figure 2. A basis of the place-invariants of the net is the following (all of them corresponding to state machines):

$$\begin{array}{ll} s_1 + s_4 = 1 & s_2 + s_3 = 1 \\ s_5 + s_8 = 1 & s_6 + s_7 = 1 \end{array}$$

Interestingly, these invariants can be used to obtain a superset of the state space. By using the encoding proposed in section 4.1 with four boolean variables ($x_1 \dots x_4$) the characteristic function

$$\hat{S}_0 = 1$$

would be obtained. Note that this is the characteristic function of the 16 states depicted in Figure 2. Even though the state space is larger, the characteristic function is simpler than (3).

Modulo-invariants provide a new invariant

$$I = s_1 + s_2 + s_5 + s_6 \equiv 0 \pmod{2}$$

indicating that the token count of the places in the invariant must remain 0 modulo 2. As an example, the marking [1357] fulfils the invariant since

$$I = 2 \equiv 0 \pmod{2}$$

whereas [1257] does not since

$$I = 3 \not\equiv 0 \pmod{2}$$

Invariant I removes all shadowed states of Figure 2 from the reachability set. The characteristic function of the markings fulfilling the modulo-invariant corresponds to the boolean equation (3). In general, modulo-invariants provide more stringent conditions for reachability than the state equation. In our example, they are able to obtain the exact state space.

5 Putting everything together

The methods described in the previous section can be combined in the same verification framework. This is illustrated in Figure 4.

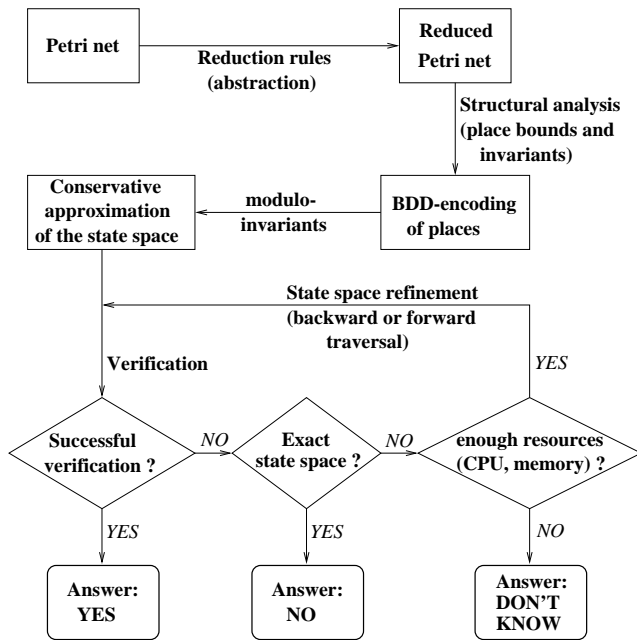


Figure 4: Putting everything together

Reduction rules can be applied at the earliest steps of the verification to simplify the structure of the Petri net. Structural methods based on the state equation and place invariants can be used to derive an efficient encoding of the markings. Next, the characteristic function of the potentially reachable state space can be derived from the modulo-invariants of the net.

Finally a cyclic verification process starts. This process completes when some of the following conditions holds:

- The verified property conservatively holds for an approximate state space.
- The exact state space has been reached. Then the result of the verification (either positive or negative) is also exact.
- The verified property does not hold in an approximate state space and there are no more resources (time and/or memory) to obtain a further refinement of the state space. The answer to the verification process is “don’t know” and the designer must conservatively assume that is negative.

The successive refinements can be obtained by applying one or several steps of the backward state elimination strategy (4). In case a fix point is reached, a forward traversal from m_0 must be performed using the information about unreachable states as “don’t

care” conditions for the manipulation of boolean characteristic functions.

6 Conclusions

The results on the structural theory of Petri nets make this formalism attractive for the specification and verification of concurrent systems. A Petri net-based verification framework can also be applied to other event-based models, such as process algebras, from which Petri nets can be derived, e.g. by syntax-directed translation techniques.

This paper has presented a strategy to integrate structural and symbolic methods in the same model-checking verification framework, thus taking advantage of the efficient algorithms devised at each domain.

Recent research by Esparza and Melzer [6] has proposed to perform conservative verification with the information provided by transition-invariants. The utilization of constraint programming [10] to derive “realizable” transition-invariants results in an efficient strategy to fight against the state explosion problem. The integration of constraint programming in model checking techniques seems to deserve further investigation.

Acknowledgments

I wish to thank Michael Kishinevsky, Luciano Lavagno and Enric Pastor for numerous discussions on the topics presented in this paper.

References

- [1] A. Arnold. *Finite transition systems: semantics of communicating systems*. Prentice Hall, 1994.
- [2] R. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.
- [3] G. Cabodi and P. Camurati. Symbolic FSM traversals based on the transition relation. *IEEE Trans. on Computer-Aided Design*, 16(5):448–457, May 1997.
- [4] O. Coudert, C. Berthet, and J. C. Madre. Verification of sequential machines using boolean functional vectors. In L. Claesen, editor, *Proc. IFIP Int. Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, Leuven, Belgium, November 1989.
- [5] J. Desel, K.-P. Neuendorf, and M.-D. Radola. Proving nonreachability by modulo-invariants. *Theoretical Computer Science*, 153:49–64, 1996.

- [6] J. Esparza and S. Melzer. Model checking LTL using constraint programming. In *18th International Conference on Application and Theory of Petri Nets*, pages 1–20, June 1997.
- [7] M. Hack. *Analysis of production schemata by Petri nets*. PhD thesis, MIT, 1972.
- [8] R. Kannan and A. Bachem. Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. *SIAM J. Comput.*, 8(4):499–577, November 1979.
- [9] A. Kovalyov and J. Esparza. A polynomial algorithm to compute the concurrency relation of free-choice signal transition graphs. In *Proc. of the International Workshop on Discrete Event Systems (WODES)*, pages 1–6, August 1996.
- [10] K. McAloon and C. Tretkoff. *Optimization and Computational Logic*. John Wiley & Sons, 1996.
- [11] K.L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.
- [12] T. Murata. Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
- [13] E. Pastor and J. Cortadella. Efficient encoding schemes for symbolic analysis of Petri nets. In *Proc. of the Conference on Design, Automation and Test in Europe (DATE)*, March 1998.
- [14] E. Pastor, O. Roig, J. Cortadella, and R. Badia. Petri net analysis using boolean manipulation. In *15th International Conference on Application and Theory of Petri Nets*, pages 416–435, June 1994.
- [15] D. Peled. Combining partial order reductions with on-the-fly model-checking. *Formal Methods in System Design*, 8:39–64, 1996.
- [16] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Bonn, Institut für Instrumentelle Mathematik, 1962. (technical report Schriften des IIM Nr. 3).
- [17] M. Silva. *Las Redes de Petri: en la Automática y la Informática*. AC, 1985. (in Spanish).
- [18] A. Valmari. Stubborn sets for reduced state space generation. In *10th International Conference on Application and Theory of Petri Nets*, pages 1–22, June 1989.